Towards Implicit Resistive Magnetohydrodynamics with Local Mesh Refinement[1]

Michael Pernice[2] Bobby Philip[2] Luis Chacón[3]
Los Alamos National Laboratory
Los Alamos, NM 87544

Solution Methods for Large-scale Nonlinear Problems
Livermore, CA
August 6–8, 2003

[1]This work was performed under the auspices of the U.S. Department of Energy by Los Alamos National Laboratory under contract W-7405-ENG-36. Los Alamos National Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness. LAUR 03-3541

[2]Computer and Computational Sciences Division
[3]Theoretical Division

Los Alamos
NATIONAL LABORATORY

Outline

- Motivation
- Resistive Magnetodynamics Model
- Physics-based Preconditioner
- Software Infrastructure
- Preconditioner Components
- Verification Efforts
- Observations

Motivation

- Models of resistive MHD contain multiple length and time scales.
  - Solutions often have highly localized spatial features.
  - Implicit timestepping schemes are needed.
    - Explicit time stepping requires $\Delta t \lesssim \mathcal{O}(\Delta x^2)$ when diffusion/Hall effects are significant.
    - Semi-implicit methods allow $\Delta t \lesssim \mathcal{O}(\Delta x)$.
      - ★ Accuracy often requires somewhat smaller values.
    - Implicit time steps are constrained only by accuracy requirements.
      - ★ Time steps closer to time scales of interest are possible.
      - ★ Fast solution of large-scale systems of nonlinear equations is necessary for this to be competitive.
- Important applications exist in space weather and fusion simulation.

Reduced Resistive MHD Model

Two-dimensional incompressible plasma:

$$\begin{aligned} \Delta\Phi &= \omega \\ (\partial_t + \mathbf{u}\cdot\nabla - \mu\Delta)\,\Psi + E_0 &= 0 \qquad \text{on } \Omega = [0, L]\times[0,1] \\ (\partial_t + \mathbf{u}\cdot\nabla - \nu\Delta)\,\omega + S_\omega &= \mathbf{B}\cdot\nabla J \end{aligned}$$

where

$$J = \Delta\Psi \qquad \mathbf{u} = \vec{k}\times\nabla\Phi = \begin{pmatrix} -\Phi_y \\ \Phi_x \end{pmatrix} \qquad \mathbf{B} = \vec{k}\times\nabla\Phi = \begin{pmatrix} -\Psi_y \\ \Psi_x \end{pmatrix}$$

Periodic boundary conditions in x and homogeneous Dirichlet boundary conditions in y.

Equilibrium sources are chosen to balance prescribed initial conditions:

$$E_0 = \mu\Delta\Psi_0, \qquad S_\omega = \nu\Delta\omega_0.$$

Semi-discretization in Time

Crank-Nicolson discretization in time is used. Given a solution $(\Phi^{(n)}, \Psi^{(n)}, \omega^{(n)})$ at time level n, the time-advanced solution is the root of

$$
\begin{aligned}
F(\Phi, \Psi, \omega) \quad = \quad & \begin{pmatrix} \Delta\Phi - \omega \\ \frac{\Psi}{\Delta t} + \theta \left(\nabla \cdot (\mathbf{u}\Psi) - \mu \Delta \Psi \right) \\ \frac{\omega}{\Delta t} + \theta \left(\nabla \cdot (\mathbf{u}\omega) - \nu \Delta \omega - \mathbf{B} \cdot \nabla J \right) \end{pmatrix} \\
& + \begin{pmatrix} 0 \\ -\frac{\Psi^{(n)}}{\Delta t} + (1-\theta) \left(\nabla \cdot (\mathbf{u}^{(n)} \Psi^{(n)}) - \mu \Delta \Psi^{(n)} \right) \\ -\frac{\omega^{(n)}}{\Delta t} + (1-\theta) \left(\nabla \cdot (\mathbf{u}^{(n)} \omega^{(n)}) - \nu \Delta \omega^{(n)} - \mathbf{B}^{(n)} \cdot \nabla J^{(n)} \right) \end{pmatrix} \\
& + \begin{pmatrix} 0 \\ \mu \Delta \Psi_0 \\ \nu \Delta \omega_0 \end{pmatrix} .
\end{aligned}
$$

Preconditioner

Every nonlinear iteration of every time step requires solution of a system of linear equations

$$F'(\Phi_k^{(n+1)}, \Psi_k^{(n+1)}, \omega_k^{(n+1)}) \begin{pmatrix} \delta\Phi \\ \delta\Psi \\ \delta\omega \end{pmatrix} = -F(\Phi_k^{(n+1)}, \Psi_k^{(n+1)}, \omega_k^{(n+1)}).$$

The task is to find a preconditioner $M \approx F'(\Phi_k^{(n+1)}, \Psi_k^{(n+1)}, \omega_k^{(n+1)})$ such that

$$M \begin{pmatrix} z_\Phi \\ z_\Psi \\ z_\omega \end{pmatrix} = \begin{pmatrix} r_\Phi \\ r_\Psi \\ r_\omega \end{pmatrix}$$

is "easy" to solve.

Semi-implicit Preconditioner

Chacón, Knoll and Finn[4] derived an effective physics-based preconditioner based on solving

$$\begin{pmatrix} \mathcal{L}_\mu & -\theta \mathbf{B}_0 \cdot \nabla \\ -\theta \mathbf{B}_0 \cdot \nabla & \mathcal{L}_\nu \end{pmatrix} \begin{pmatrix} \delta\Psi \\ \delta\Phi \end{pmatrix} = \begin{pmatrix} r_\Psi \\ \nabla^{-2}[r_\omega - \mathcal{L}_\nu r_\Phi] \end{pmatrix}$$

for $\delta\Psi$ and $\delta\Phi$, followed by solving

$$\mathcal{L}_\nu \delta\omega = r_\omega + \theta(\nabla \cdot (\mathbf{B}_0 \delta J) + \nabla \cdot (\delta \mathbf{B} J_0) + (\vec{k} \times \nabla\omega_0) \cdot \nabla\delta\Phi).$$

Here, $\mathcal{L}_\nu = \frac{1}{\Delta t} + \theta\,(\mathbf{u}_0 \cdot \nabla - \nu\Delta)$ and $\mathcal{L}_\mu = \frac{1}{\Delta t} + \theta\,(\mathbf{u}_0 \cdot \nabla - \mu\Delta)$.

[4] This workshop, Friday at 11; see also JCP, **178**, 2002.

Reformulation as Stationary Method

$$\underbrace{\begin{pmatrix} \mathcal{L}_\mu & -\theta\mathbf{B}_0\cdot\nabla \\ -\theta\mathbf{B}_0\cdot\nabla & \mathcal{L}_\nu \end{pmatrix}}_{\mathcal{P}} = \underbrace{\begin{pmatrix} \mathcal{L}_\mu & -\theta\mathbf{B}_0\cdot\nabla \\ -\theta\mathbf{B}_0\cdot\nabla & \mathcal{D}_\nu \end{pmatrix}}_{\mathcal{M}} - \begin{pmatrix} 0 & 0 \\ 0 & \mathcal{D}_\nu - \mathcal{L}_\nu \end{pmatrix}$$

$$\begin{pmatrix} \delta\Psi \\ \delta\Phi \end{pmatrix} \leftarrow \begin{pmatrix} \delta\Psi \\ \delta\Phi \end{pmatrix} + \mathcal{M}^{-1}\left(\begin{pmatrix} r_\Psi \\ \nabla^{-2}[r_\omega - \mathcal{L}_\nu r_\Phi] \end{pmatrix} - \mathcal{P}\begin{pmatrix} \delta\Psi \\ \delta\Phi \end{pmatrix}\right)$$

$$\mathcal{M} = \begin{pmatrix} 1 & -\theta\mathbf{B}_0\cdot\nabla\mathcal{D}_\nu^{-1} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \mathcal{L}_\mu - \theta^2\nabla\cdot\mathbf{B}_0\mathcal{D}_\nu^{-1}\mathbf{B}_0^t\nabla & 0 \\ -\theta\mathbf{B}_0\cdot\nabla & \mathcal{D}_\nu \end{pmatrix}.$$

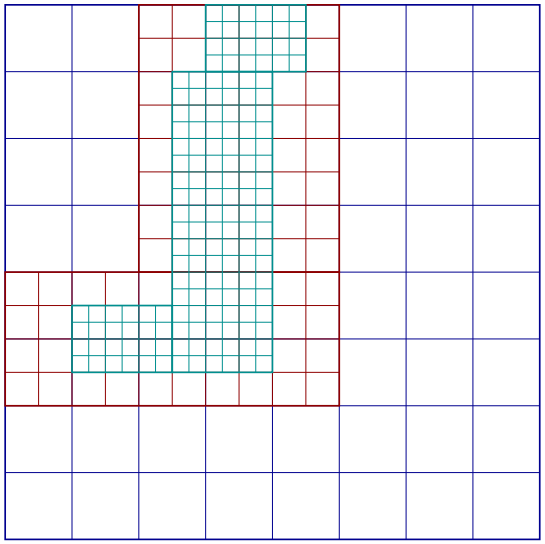
Preconditioner Summary

Procedure applyRMHDPreconditioner:

SET $R_\Psi = r_\Psi$.
SET $R_\Phi = \nabla^{-2}[r_\omega - \mathcal{L}_\nu r_\Phi]$.
SET $\delta\Psi_0 = \delta\Phi_0 = 0$.
SEMI-IMPLICIT ITERATION: FOR $m = 0, \ldots, max\_iters$ {
  SET $f_\Psi = R_\Psi + \theta \mathbf{B}_0 \cdot \nabla \mathcal{D}_\nu^{-1} R_\Phi$.
  SOLVE $\left(\mathcal{L}_\mu - \theta^2 \nabla \cdot \mathbf{B}_0 \mathcal{D}_\nu^{-1} \mathbf{B}_0^t \nabla\right) \delta\Psi = f_\Psi$.
  SET $f_\Phi = R_\Phi + \theta \mathbf{B}_0 \cdot \nabla \delta\Psi$.
  SET $\delta\Phi = \mathcal{D}_\nu^{-1} f_\Phi$.
  $\delta\Psi_{m+1} = \delta\Psi_m + \delta\Psi$, $\delta\Phi_{m+1} = \delta\Phi_m + \delta\Phi$.
  UPDATE $R_\Psi = r_\Psi - \mathcal{L}_\mu \delta\Psi_{m+1} + \theta \mathbf{B}_0 \cdot \nabla \delta\Phi_{m+1}$.
  UPDATE $R_\Phi = \nabla^{-2}[r_\omega - \mathcal{L}_\nu r_\Phi] - \mathcal{L}_\nu \delta\Phi_{m+1} + \theta \mathbf{B}_0 \cdot \nabla \delta\Psi_{m+1}$.
}
SOLVE FOR $\delta\omega$.

Structured Adaptive Mesh Refinement

Structured adaptive mesh refinement (SAMR) represents a locally refined mesh as a union of logically rectangular meshes.

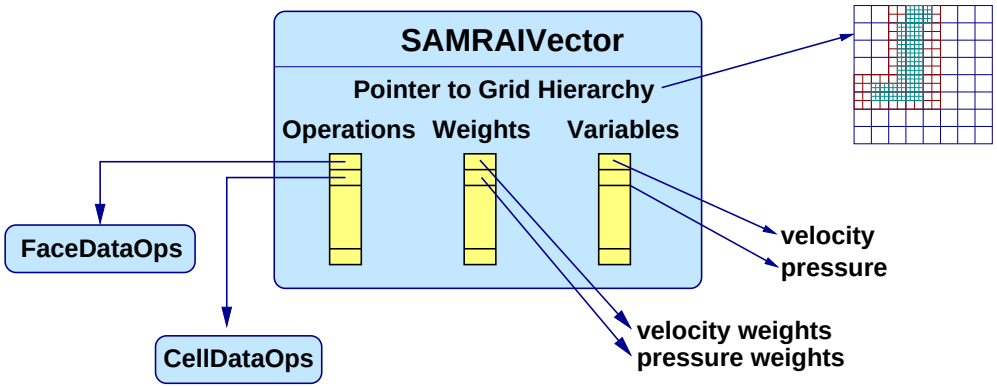


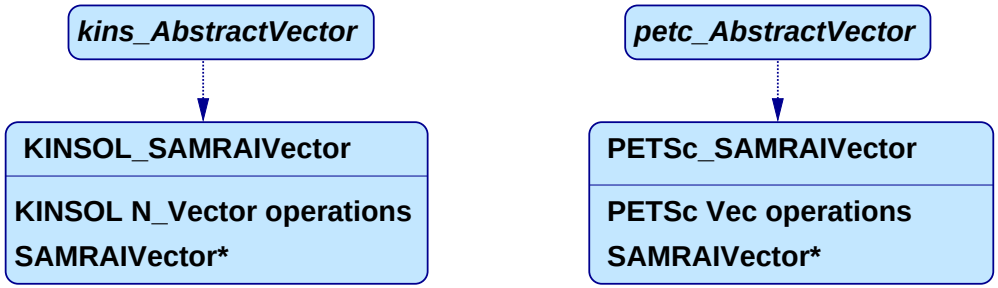
- The mesh is organized as a hierarchy of nested *refinement levels*.
- Each refinement level defines a region of uniform resolution.
- Each refinement level is the union of logically rectangular *patches*.

Software Infrastructure

SAMRAI is used to manage the complexity associated with a dynamic, locally refined grid.

Interoperability to solver packages is enabled through use of software wrappers, which allow the solvers to operate directly on grid-based data *with no copy overhead*.





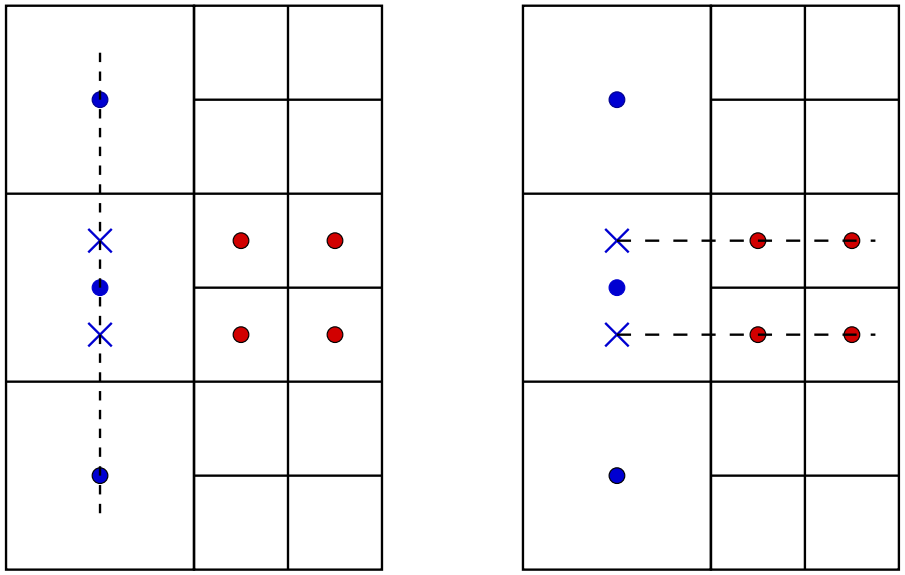
Los Alamos NATIONAL LABORATORY

Requirements for Extension to SAMR

With software that implements inexact Newton methods on SAMR grids, the task is to write the required methods for function evaluation and management of the preconditioner.

- Nonlinear function evaluation.
  - All spatial discretizations must properly account for changes in resolution.
- Setup and apply the preconditioner.
  - Generalize preconditioner components from multigrid solvers to Fast Adaptive Composite grid (FAC) solvers:
    - Poisson solver
    - convection-tensor diffusion solver
    - convection-diffusion solver

Spatial Discretization at Changes in Resolution



This must be handled properly for each of:

$$\nabla \qquad \nabla \times \qquad \mathbf{u} \cdot \nabla \qquad \mathbf{B} \cdot \nabla \qquad \nabla \cdot \overline{\overline{\mathbf{D}}} \nabla$$

FAC

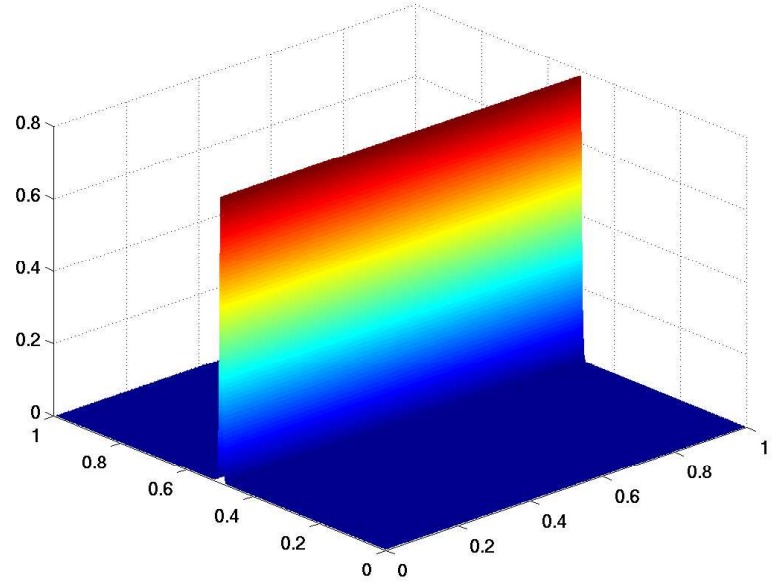
Procedure FAC($\underline{h}$, $f^{\underline{h}}$, $u^{\underline{h}}$):

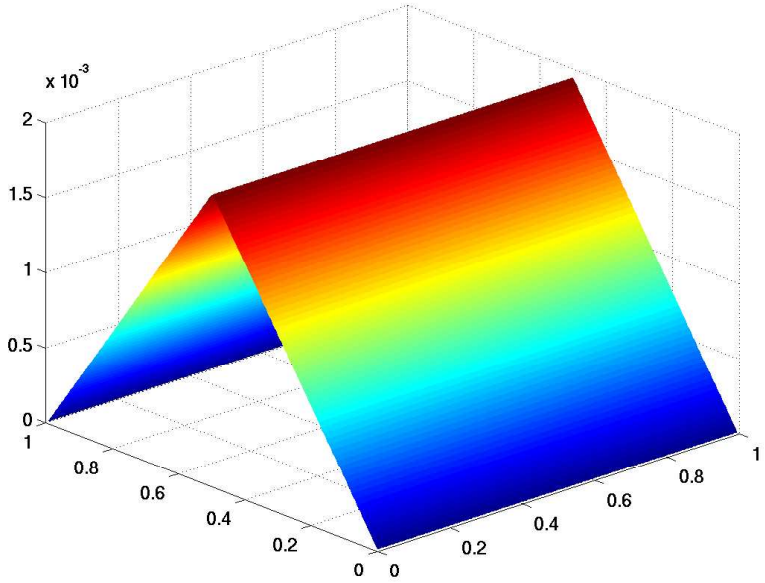
If $\underline{h} = \{h_c\}$, solve $L^{h_c} u^{h_c} = f^{h_c}$ and return.
Set $f^h = I_{\underline{h}}^h (f^{\underline{h}} - L^{\underline{h}} u^{\underline{h}})$.
Solve/Smooth $L^h u^h = f^h$.
Correct $u^{\underline{h}} = u^{\underline{h}} + I_h^{\underline{h}} u^h$.
Set $u^{\underline{2h}} = 0$, $f^{\underline{2h}} = I_{\underline{h}}^{\underline{2h}} (f^{\underline{h}} - L^{\underline{h}} u^{\underline{h}})$.
$u^{\underline{2h}} = \mathsf{FAC}(\underline{2h}, f^{\underline{2h}}, u^{\underline{2h}})$.
Correct $u^{\underline{h}} = u^{\underline{h}} + I_{\underline{2h}}^{\underline{h}} u^{\underline{2h}}$.
Set $f^h = I_{\underline{h}}^h (f^{\underline{h}} - L^{\underline{h}} u^{\underline{h}})$.
Solve/Smooth $L^h u^h = f^h$.
Correct $u^{\underline{h}} = u^{\underline{h}} + I_h^{\underline{h}} u^h$.

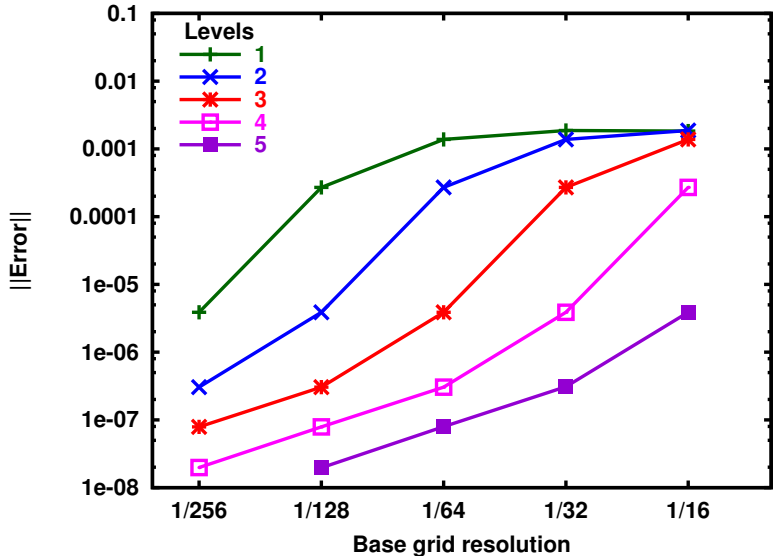
Implementation of Preconditioner Components

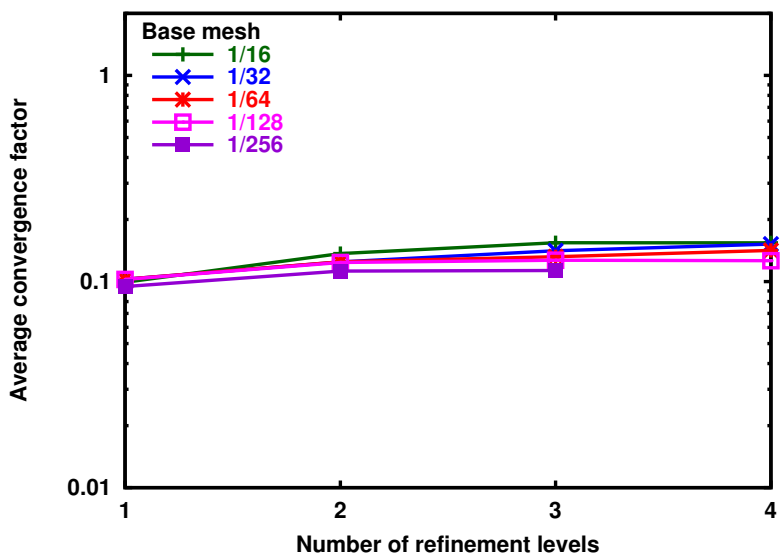
- Solves on the coarsest level using one V-cycle of the smg solver from *hypre*.
- Simple red-black point Gauß-Seidel smoothing on finer levels.
- Interlevel transfers use built-in geometric interpolation from SAMRAI.

Verification of Poisson solver



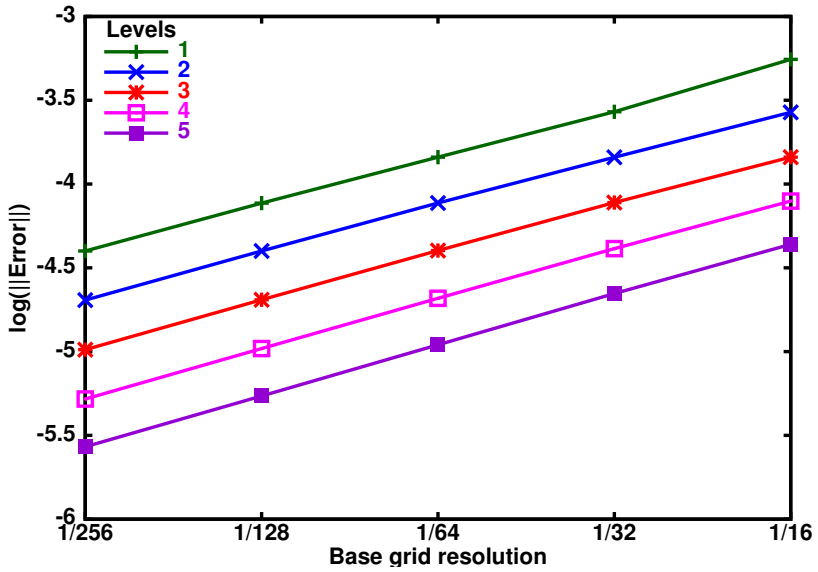


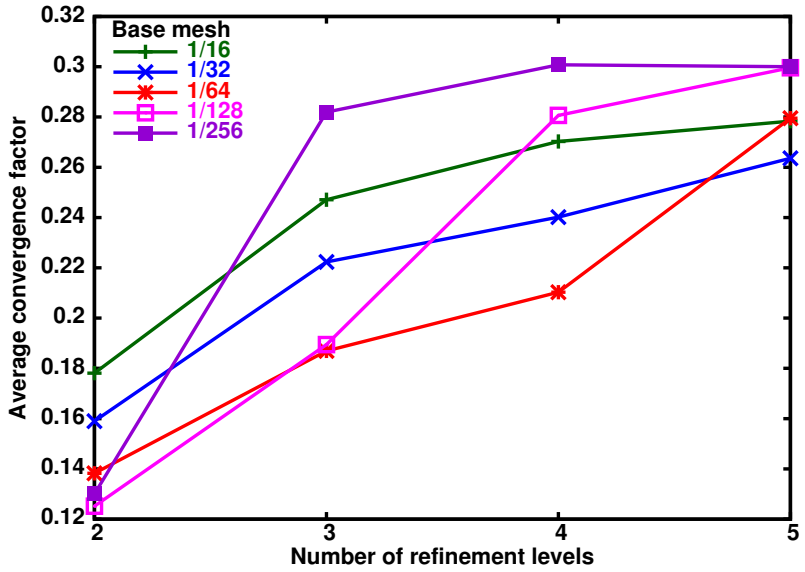




Los Alamos NATIONAL LABORATORY

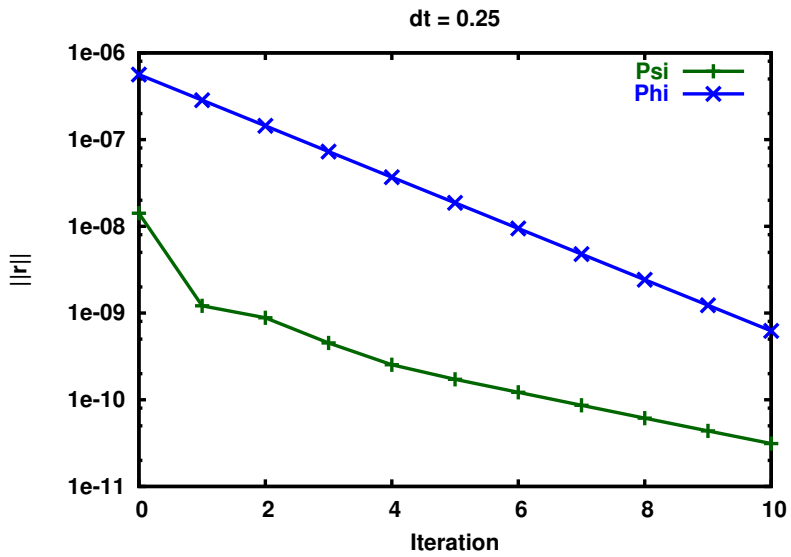
Verification of Convection-Tensor Diffusion Solver

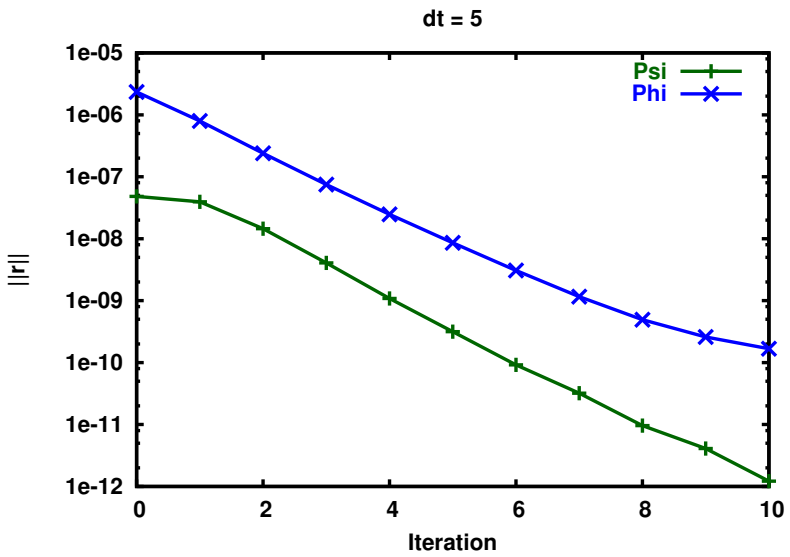


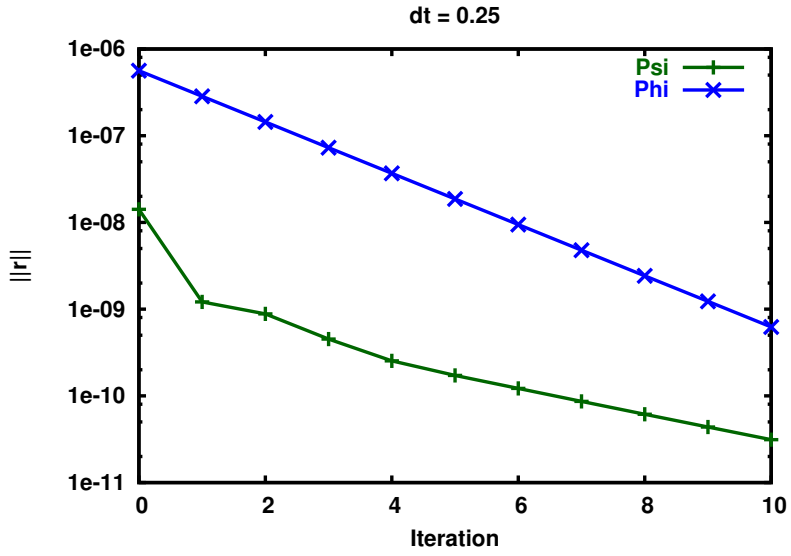


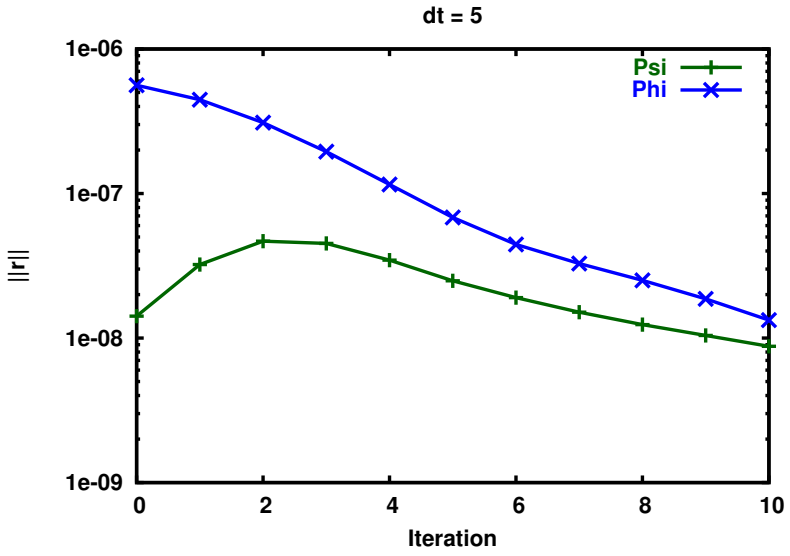
Los Alamos NATIONAL LABORATORY

Verification of Semi-Implicit Iteration



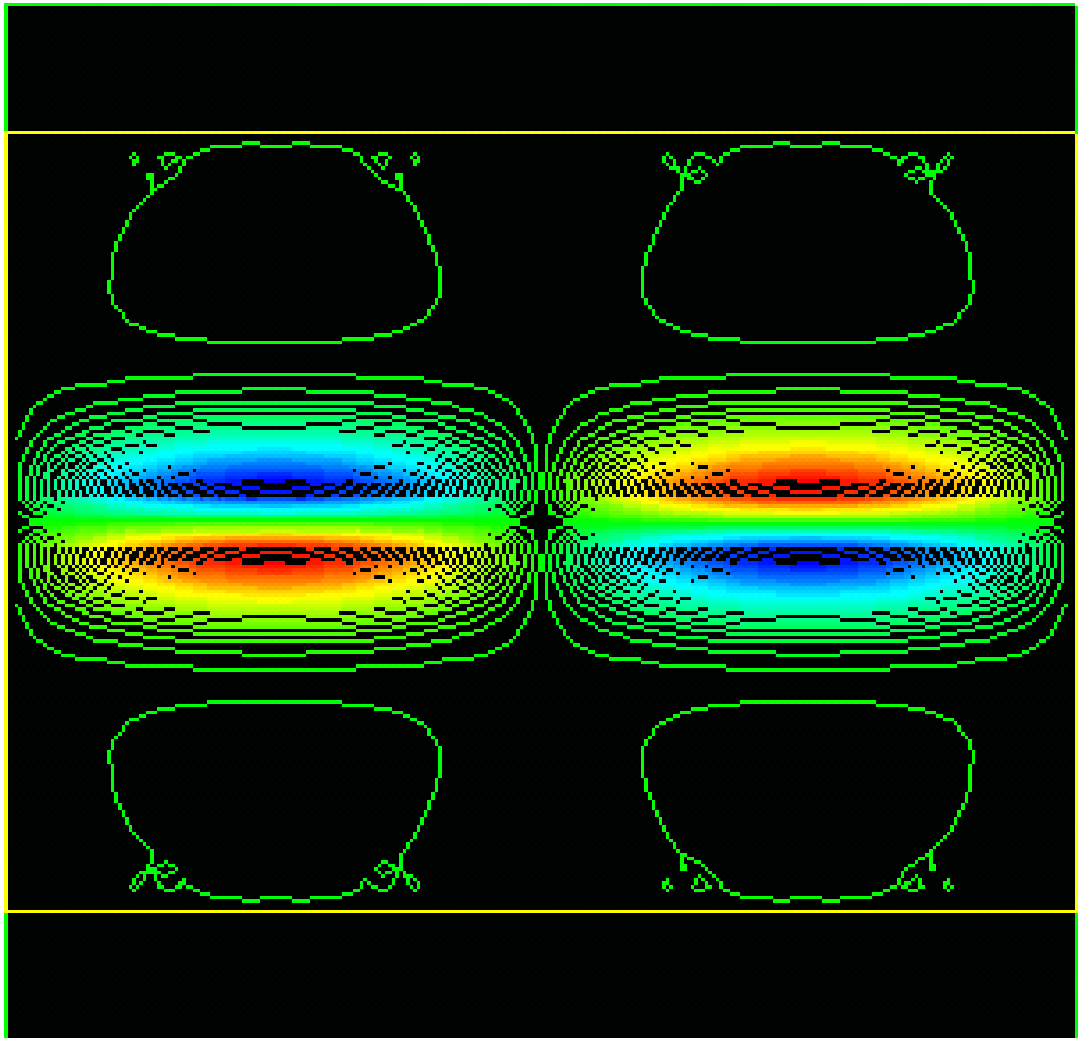






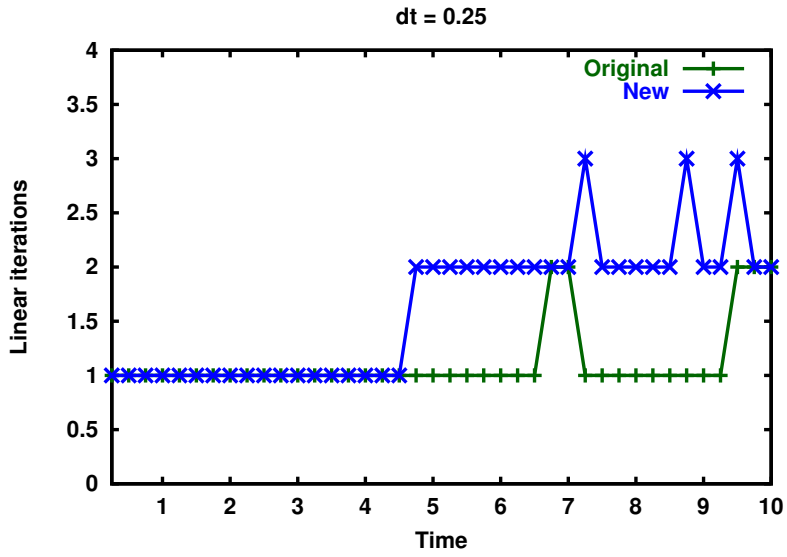
Los Alamos NATIONAL LABORATORY

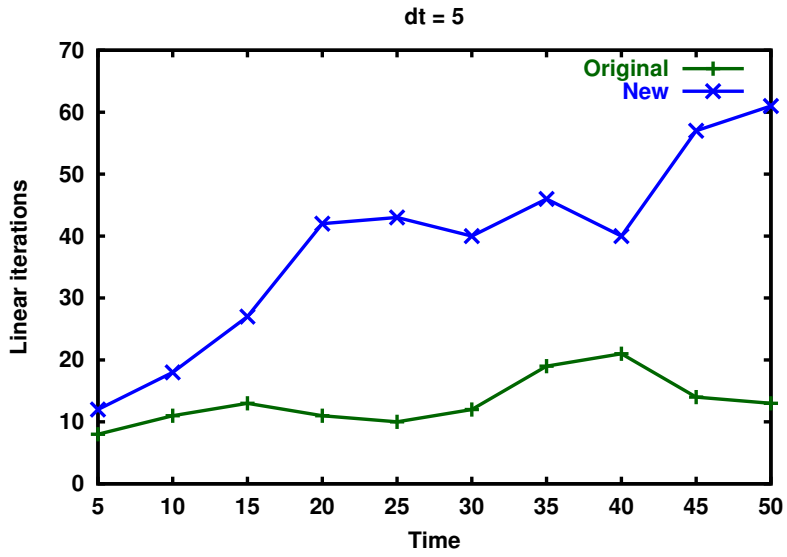
Verification of Function Evaluation



Los Alamos
NATIONAL LABORATORY

Verification of Single Level Performance





Los Alamos NATIONAL LABORATORY

Observations

- A preconditioner interface that consists of four functions
  1. `createPreconditioner`
  2. `setupPreconditioner`
  3. `applyPreconditioner`
  4. `destroyPreconditioner`

  has proven to be quite useful and provides all needed flexibility.
- Surprisingly little existing code was reused.
- A component-based approach to building complex applications can provide both flexibility and needed high-level abstractions to simplify implementation.
  - The right level of granularity is hard to find.
  - Comprehensive testing is necessary, but *not sufficient*.
  - *No monolithic components.*
- Decoupling solver abstractions from other abstractions (grid management, geometry management, data transfer) can lead to wider applicability and eliminates most copy operations.